G. Rudolph and T. Jansen and S. Lucas and C. Poloni and N. Beume *Eds.*, PPSN 2008.
13-17 Sept, p1061–1070, LNCS 5199, Springer.

Evolving Regular Expressions for GeneChip Probe Performance Prediction

W. B. Langdon and A. P. Harrison

Departments of Mathematical, Biological Sciences and
Computing and Electronic Systems, University of Essex, UK

Abstract. Affymetrix High Density Oligonuclotide Arrays (HDONA) simultaneously measure expression of thousands of genes using millions of probes. We use correlations between measurements for the same gene across 6685 human tissue samples from NCBI's GEO database to indicated the quality of individual HG-U133A probes. Low concordance indicates a poor probe. Regular expressions can be data mined by a Backus-Naur form (BNF) context-free grammar using strongly typed genetic programming written in `gawk` and using `egrep`. The automatically produced motif is better at predicting poor DNA sequences than an existing human generated RE, suggesting runs of Cytosine and Guanine and mixtures should all be avoided.

1 Introduction

Typically Affymetrix GeneChips (e.g. HG-U133A) measure gene expression at at least eleven points along the gene. Individual measurements are given by short (25 base) DNA sequences, known as probes. These are complementary to corresponding locations in genes. Being complementary, the gene product (messenger RNA) preferentially binds to the probe. Cf. left hand side of Figure 1. Half a million probes are placed on a glass slide in a square grid pattern. A fluorescent dye is used to measure how much mRNA is bound to each probe.

While nothing is simple in Biology, to a first approximation, the amount of mRNA produced by a gene should be the same no matter which part of the mRNA molecule is bound to a probe. Affymetrix groups probes into probesets. Each probeset targets a gene. Therefore probe measurements for the same probeset should be correlated. Figure 1 (right) shows the 110 correlations for a probeset as a "heatmap" (yellow/lighter corresponds to greater consistency between pairs of probes).

There are several biological reasons which might lead to probes on the same gene giving consistently unrelated readings. (Alternative splicing, alternative polyadenylation and 3'-5' degradation, come to mind [11].) However these do not explain all of the many cases of poor correlation. In [9] we found some technological reasons. In particular, [9] showed that probes containing a large ratio of Guanine (G) to Adenosine (A) bases are likely to perform badly. Subsequently we have found that runs of Gs (which will tend to have a high G/A ratio) also

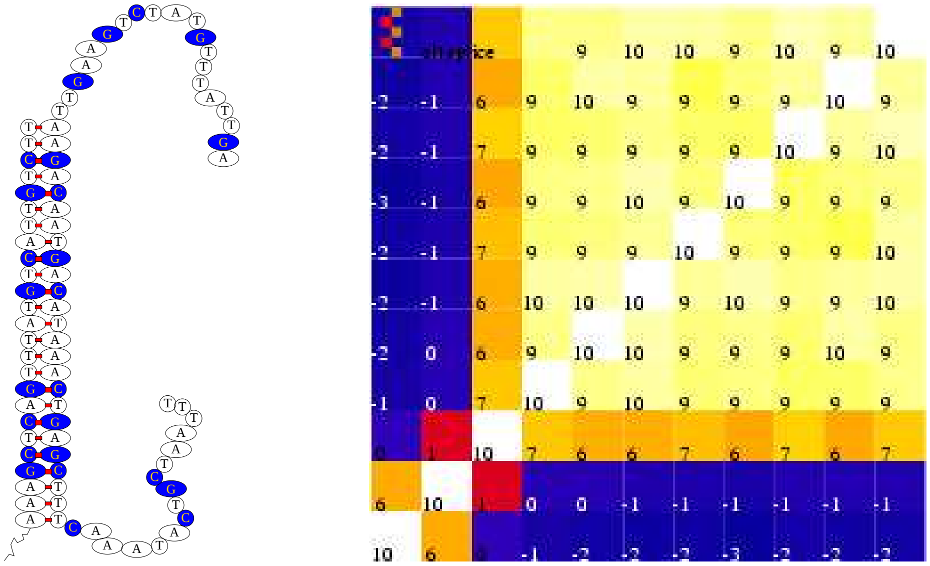

Fig. 1. Left:Schematic of an Affymetrix probe (vertical) bound with complementary target sequence (right). Right: Correlation coefficients ($\times 10$) between 11 probes for gene "S100 calcium binding protein A11" S100A11. Nine of the probes are correlated but PM_1 and PM_2 (bottom 2 rows and 2 left) are not.

tend to indicate problem probes [25]. This has lead us to ask if there are other *sequences* which might indicate dodgy probes.

The next section will briefly describes the research background, whilst Section 3 will describe the preparation of datasets containing the correlation coefficients and probe sequences. Section 4 describes the evolutionary algorithm used to create regular expressions for `egrep`. Section 5 describes how well genetic programming does. Section 6 uses the evolved motif to suggest potential physical explanations for poor probes. Finally, in Section 7 we conclude that several regular expressions, e.g. `G(G|C){4}`, in additions to `GGGG`, can be used together to locate poor probes.

2 Grammars and Evolutionary Computation

Existing research on using grammars to constrain the evolution of programs can be broadly divided in two: Grammatical evolution [21] based largely in Ireland and work in the far east by Wigham [26; 27], Wong [28] and McKay [17].

There is quite a body of work on using evolution to induce formal grammars. E.g. Nikolaev tackled the Tomita regular expression benchmarks [20]. GP has also evolved context free grammars [14]. Cetinkaya used grammatical evolution to create regular expression for processing HTML [6].

Ross induced stochastic regular expressions from a number of grammars to classify proteins from their amino acid sequence [24]. Typically his grammars had eight alternatives. In Stockholm regular expressions have been evolved to search for similarities between proteins, again based on their amino acid sequences [7]. Whilst Brameier in Denmark used amino acids sequences to predict the location of proteins by applying a multi-classifier [16] linear GP based approach [4] (although this can be done without a grammar [15]). A similar technique has also been applied to study microRNAs [5].

The next section describes our single stage strongly typed tree based GP being applied to an important DNA problem: explaining why some DNA sequences in wide spread commercial use make poor GeneChip probes.

3 Preparation of Training Data

Previously we had down loaded thousands of experiments from NCBI's GEO [2], normalised them, excluded spatial defects and calculated the correlation between millions of pairs of probes [13; 9; 12]. To exclude genes which are never expressed, we selected probesets where ten or more non-overlapping probe pairs had correlations of 0.8 or more. For each probe we use the median value of all 10 of its correlations with other members of its probeset (excluding those it overlaps). This gave 4118 probesets, which were evenly split into three to provide independent training, test and validation data.

Previously we found the "mismatch" probes were often poorly correlated with other measurements for the same gene [9]. Since this is known, we excluded them from this study.

As Figure 2 (left) shows correlation coefficients cover a wide range. Since we are using correlation only as an indication of how well a probe is working we decided to exclude the middle values from training and instead use probe pairs that were highly correlated (≥ 0.8) or were very poorly correlated (≤ 0.3). Of the 15 092 available training examples, there are 7 832 probes highly correlated with the rest of their probeset but only 583 poorly correlated. To avoid unbalanced training sets, every generation all 583 negative examples are used and 583 positive examples are randomly chosen from the 7 832 positive examples.

4 Evolving Regular Expression Motifs

4.1 BNF grammar of Regular Expression

The BNF grammar used (cf. Figure 3) is an extension of that given by Cameron `http://www.cs.sfu.ca/people/Faculty/cameron/Teaching/384/99-3/regexp-plg.html` In particular, matching the beginning of strings (`^`) and the `{n,m}` form of Kleen closure, are also supported. The BNF has been customised for DNA strings. (I.e. `<char>` need only be `A` `C` `G` and `T`). Since various combinations of the start of string symbol, null strings and Kleen closure cause `egrep` to loop, care has been taken to ensure that the new BNF does not permit null strings after `^`.

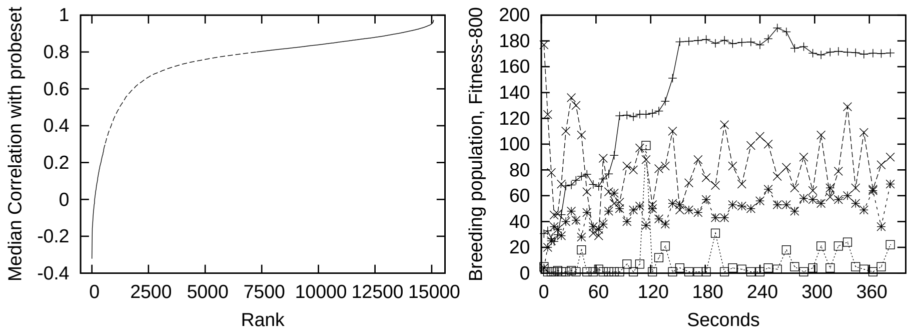

Fig. 2. Training data. Probes with intermediate values ($0.3 \dots 0.8$) are not used.

Right: Evolution of breeding population (best 200 of 1000) of regular expressions to find poor GeneChip probes. Each generation the positive training cases are replaced leading to fluctuations in the measured best fitness (*). However diversity remains high and there are usually few individuals with the same highest fitness (□). In this run the number of distinct phenotypes ($\times$) (i.e. `egrep` search strings) is almost identical to the number of distinct genotypes. Bloat is limited (+), apparently by the tree depth limit (17). However the system slows down by ($\approx \frac{1}{2}$) as evolution proceeds.

Brameier and Wiuf suggests that the traditional `*` and `+` form of Kleen closure are not suitable for bioinformatic applications [5]. Instead they recommend the `{n,m}` form which explicitly defines both lower (`n`) and upper (`m`) limits on the number of times the preceeding symbol must occur. However both `{n,m}` and traditional Kleen closures are used by evolved solutions. To avoid `mutation.awk` seeing "Hamming cliffs", the integer quantifiers used in the `{n,m}` are Gray coded [1]. Similarly the syntax groups together the chemically more similar Pyrimidines (`T` and `C`) and Purines (`A` and `G`).

Our system supports full positive integer values for `minmaxquantifier`, however even modest values can lead `egrep` to hang the computer. Therefore `n` and `m` are limited to 1-9. Finally `egrep` rejects `{n,m}` if `m<n`. This is handled by a semantic rule which removes `,m` from the phenotype when `m` is less than `n`.

4.2 Simplifying the BNF for use by Evolutionary Algorithms

For simplicity, the BNF is written so that grammar rules are either simple substitution rules (e.g. `<minmaxquantifier>`), rules with exactly two options (e.g. `<RE>`) or terminals (e.g. `"*"` and `T`). In BNF terms, a terminal is a symbol which cannot be substituted in the grammar. Therefore, unlike the BNF rules, it becomes part of the `egrep` regular expression. The simple substitution rules do not have any element of choice. They, like terminals, cannot be chosen as crossover points or targets for mutation. Their principle use is to enable the rules with options to be kept simple.

The binary choice rules are the active parts of the syntax. As they are always binary, each sentence recognised by the BNF has an equivalent binary string.

```
<start>  ::=  <RE>
<RE>  ::=  <union> | <simple-RE>
<union>  ::= <RE> "|" <simple-RE>
<simple-RE>  ::=  <concatenation> | <basic-RE>
<concatenation>  ::= <simple-RE> <basic-RE>
<basic-RE>  ::= <RE-kleen> | <elementary-RE>
<RE-kleen>::= <minmaxquantifier> | <kleen>
<kleen>::= <star> | <plus>
<star>  ::= <elementary-RE2> "*"
<plus>  ::= <elementary-RE2> "+"
<minmaxquantifier>   ::= <elementary-RE4> "{" <int> <optREint> "}"
<elementary-RE>  ::= <group> | <elementary-RE1>
<elementary-RE1>  ::= <xos> | <elementary-RE2>
<elementary-RE2>  ::= <any> | <elementary-RE3>
<elementary-RE3>::= <set> | <char>
<elementary-RE4>  ::= <group> | <elementary-RE2>
<group>  ::=  "(" <RE> ")"
<xos>  ::=  <sos> | "$"
<sos>  ::=  "^" <elementary-RE4>
<set>  ::=  <positive-set> | <negative-set>
<positive-set>  ::=  "[" <set-items> "]"
<negative-set>  ::=  "[^" <set-items> "]"
<set-items>  ::=  <set-item> | <set-items2>
<set-items2>  ::=  <set-item> <set-items>
<set-item>  ::=  <char>
<char>  ::=  <c00> | <c01>
<any>  ::=  "."
<c00>  ::=  T | C
<c01>  ::=  A | G

<optREint>  ::=  <2ndint> | $
<2ndint>  ::=  "," <int>
<int>  ::=  <d0>
#4 Bit Gray Code Encoder
<REdigit>  ::=  <d111> | <d0>
<d0>  ::=  <d00> | <d01>
<d00>  ::=  <d000> | <d001>
<d01>  ::=  <d010> | <d011>
<d000>  ::=  1
<d001>  ::=  3 | 2
<d010>  ::=  7 | 6
<d011>  ::=  4 | 5
<d111>  ::=  8 | 9
```

Fig. 3. Grammar used to specify legal regular expressions for use as `egrep` search strings for testing DNA sequences.

(A BNF sentence means an `egrep` regular expression in our case.) Each bit corresponds to a BNF rule with two options needed when parsing the sentence. The bit indicates which option should be invoked. Note that the meaning of each bit depends upon where we have reached in parsing the sentence (i.e. on the earlier bits). In traditional GP terms, this list of choices is the evolving tree. The BNF grammar acts to put labels on the choices, which constrain crossover, but it is the choices (not the BNF grammar) which is the genetic material of the evolving individual. Note, unlike grammatical evolution, strongly typed crossover respects [23] the meaning of the BNF rules.

4.3 Creating Random BNF Sentences

The initial random population is created using ramped half-and-half [8]. It may help to think of this as applying the usual genetic programming ramped half-and-half algorithm to a binary tree (of choice nodes).

We start from `<start>` and recursively follow the BNF. However when we reach a rule with options we need to choose one. As in ramped half-and-half we keep track of how deep we are nested. If we have not reached the depth needed to terminate the recursion, we randomly choose one of the options. This is the equivalent of choosing a GP function. (As with other strongly typed GPs, if a chosen route through the syntax has no further choices to be made, we may be forced to terminate a recursive branch early.)

To terminate a recursion we choose the "simpler" option. Our BNF has been written so that the simpler option is always on the right. (This is flagged by `RE` in the rule name.) If there is no "simpler" choice, the choice is made randomly. This mechanism is also used for mutating existing regular expressions.

Although this may seem complex, `gawk` (Unix' free interpreted pattern scanning and processing language) can handle populations of a million individuals.

4.4 Crossing over BNF Sentences

Creating a new sentence from two high fitness sentences is essentially subtree crossover [22] applied to the binary choice tree (cf. Section 4.1) with the addition of strong type constraints [18]. This is implemented by scanning the grammar used to create the first parent for all the rules with two options. One of these is randomly chosen. For example, suppose the first parent starts `<start> <RE> <union>` and suppose `<union>` is chosen as the crossover point. For a grammatically correct child to be produced all that is necessary is that the crossover point chosen in the second parent should also be `<union>`. (There are complications to do with depth and size limits, which we shall ignore for the time being.) Therefore the second parent is scanned to find all occurrences of `<union>`. One of them is randomly chosen to be the second crossover point. (If there are none, this crossover is aborted and another initial crossover point is chosen. If we keep failing, eventually another pair of parents is chosen.)

Crossover is based on normal GP subtree crossover, cf. [22, Figure 2.5]. The new child is created by copying the start of the first parent, excluding the subtree at the first parent's crossover point. Then genetic material from the subtree at the second parent's crossover point is added. Finally the remainder of the first parent is appended to the child. This is implemented by crossing over the binary choice trees to create a binary choice tree for the new child. Apart from issues of tree size and depth, we are guaranteed that the new binary choice tree will represent a valid sentence in the BNF grammar.

The final step is to recursively trace through the BNF grammar, obeying the binary choices. Each time an BNF terminal (except the null symbol) is encountered it is appended to the new regular expression. In order to be able to breed following generations, the new individual's genotype must also be passed to the next generation. For convenience, this is done by writing each BNF symbol, as it is encountered, to the file holding the next generation.

4.5 Evaluating the Fitness of BNF Sentences

Each generation, a command file is generated which contains a `egrep -c -v '`*RE*`'` command for each individual in the population. (*RE* is the individual's regular expression.) The command is run on a file holding the DNA sequences of the 583 probes poorly correlated with the rest of their probeset. The same command is also run on a file holding the 583 positive probes selected for use in this generation. The fitness of the regular expression is based on the difference between the number of lines in the two files which match *RE*. Expressions which either match all probes or fail to match any are penalised by subtracting 583 from their score. See also Table 1. Implementation details can be found in [12].

5 Results

By the end of the first run, with a population of 1000 (cf. Table 1 and Figure 2, right) GP had evolved a probe performance predictor (see Figure 4) equivalent to `GGGG|CGCC|G(G|C){4}|CCC`. It is obvious that it includes the previous rule (`GGGG`, [25]) but includes other possibilities. Therefore it finds more poor probes. Inevitably it will also incorrectly predict more high correlation probes are poor but the increase is more than offset by its better performance on the poor probes. See Figure 5. It has a fitness of 856. (`GGGG` has a fitness of 776.)

The confusion matrix for the evolved regular expression on the whole of the training set (including the 6677 positive middling values which GP never saw) is $\boxed{\begin{matrix} 410 & 4448 \\ 173 & 10061 \end{matrix}}$ and on the verification data: $\boxed{\begin{matrix} 448 & 4436 \\ 174 & 10045 \end{matrix}}$. (The corresponding matrices for `GGGG` are $\boxed{\begin{matrix} 195 & 479 \\ 388 & 14030 \end{matrix}}$ and $\boxed{\begin{matrix} 209 & 434 \\ 413 & 14047 \end{matrix}}$.) Unlike in many machine learning applications, there is no evidence of over fitting. Indeed the corresponding results for the test set (second matrix of each pair) are not significantly different (χ^2, 3 dof) from those on the whole training set. The evolved regular expression picks up significantly more (χ^2, 3 dof) (448 v. 209) poorly performing probes on the validation set than the human produced regular expression. Figure 6 shows

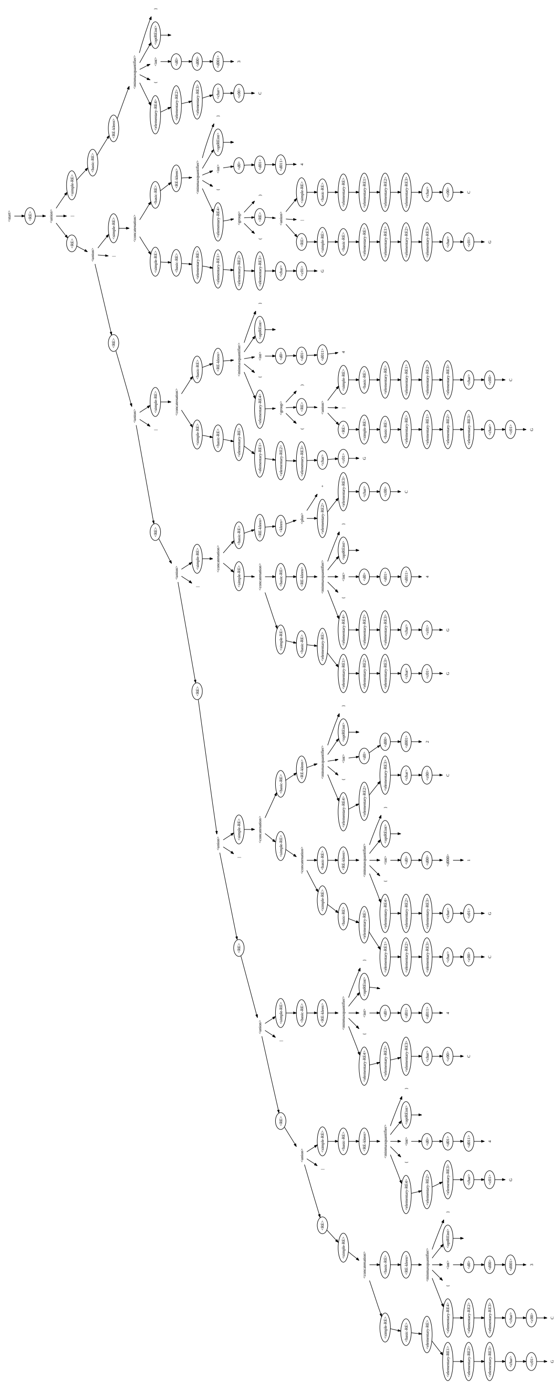

Fig. 4. Geneotype of best program in generation 50. Active choice nodes in the BNF (cf. Figure 3) are emphasised by placing them in ovals. The resulting phenotype is simply the 58 leaf nodes (excluding $), read in left to right order: `GC{3}|G{4}|C{4}|CG{1}C{2}|GG{4}C+$G(G|C){4}|G(G|C)4|C{3}`. It is equivalent to `GGGG|CGCC|G(G|C){4}|CCC`.

Table 1. Strongly Typed Grammar GP for GeneChip Correlation Prediction

Primitives:	The function and terminal sets are defined by the BNF grammar (cf. Figure 3). BNF rules with two options correspond to binary GP functions. The rest of the BNF grammar correspond to GP terminals.
Fitness:	true positives+true negatives. (I.e. proportional to the area under the ROC curve or Wilcox statistic [10].) Less large penalty if `egrep` fails or it matches all probes or none.
Selection:	(200,1000)
Initial pop:	Ramped half-and-half 3:7
Parameters:	100% subtree crossover. Max tree depth 17 (no tree size limit)
Termination:	50 generations

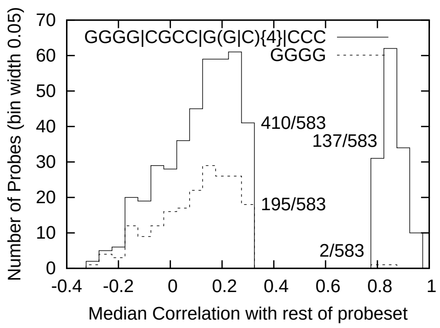

Fig. 5. Performance of evolved motif on its training data versus human generated motif (dashed).

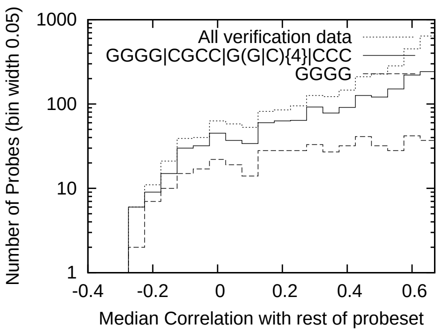

Fig. 6. Performance of evolved and human generated motifs on verification data.

the number of DNA probes matching the evolved motif against their average correlation with the rest of their probeset.

As is common in optimisation [3], almost all the time is taken by the fitness evaluation. In our case, elapse time is dominated by the command script which runs `egrep -c`. Typically this takes 8.5mS per individual. The time taken by `gawk` to process the BNF grammar, perform crossover, generate the regular expressions, etc., is negligible.

6 Discussion

Theoretical and empirical studies of GeneChips confirm that the behaviour of DNA probes tethered to a glass surface can be quite different from DNA behaviour in bulk solution. This is a new and difficult area and there are not deep pure Physics experimental results. Therefore experimental studies have concentrated on data gathered during normal operation of the chips.

Our automatically generated motif, suggests that in addition to Gs, Cs are important. Indeed the fact that only three consecutive Cs is predictive (whereas four Gs are needed) suggests that Cs are more important than Gs. It is known in GeneChips DNA C–G RNA binds more strongly than DNA G–C RNA [19]. (Both C and G bind more strongly than A and T.) We are tempted to suggest that a CCC sequence on a DNA probe can act as a nucleation site encouraging the probe to bind to GGG on RNA. Indeed the evolved motif suggests that four Gs and mixtures of five Cs and Gs might also form nucleation sites.

The sequence CCC is too short to be specific to a particular gene. GeneChips are designed on the assumption that only RNA sequences which are complementary to the full length of the probe will be stable. However studies have shown that nonspecific targets can be bound to GeneChip probes for several hours even if held only by the nucleation site. This may be why probes with quite short runs of either Cs or Gs can be poorly correlated with others designed to measure the same gene.

7 Conclusions

Access to the raw results of thousands of GeneChips (each of which costs several hundreds of pounds) makes new forms of bioinformatic data mining possible.

Millions of correlations between probes in the same probeset, which should be measuring the same gene, show wide variation. Evolution of regular expressions confirms previous work [9; 25] that the DNA sequences from which the probes themselves are formed can indicate poor probe performance. Indeed several new motifs (e.g. CCC) which predict probe quality have been automatically found.

Linux code is available via FTP `ftp://cs.ucl.ac.uk/genetic/gp-code/`

References

1. Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* OUP, 1996.

2. Tanya Barrett, *at al.* NCBI GEO: mining tens of millions of expression profiles–database and tools update. *Nucleic Acids Research*, 35:D760–D765 2007.
3. Hans-Georg Beyer. *The Theory of Evolution Strategies*. Springer, 2001.
4. Markus Brameier, Andrea Krings, and Robert M. MacCallum. NucPred predicting nuclear localization of proteins. *Bioinformatics*, 23(9):1159–1160, 2007.
5. Markus Brameier and Carsten Wiuf. Ab initio identification of human microRNAs based on structure motifs. *BMC Bioinformatics*, 8:478, 18 December 2007.
6. Ahmet Cetinkaya. Regular expression generation through grammatical evolution. In Tina Yu, ed., *GECCO-2007 workshop program*, pp2643–2646. ACM Press.
7. T. Handstad, A. J. H. Hestnes, and P. Saetrom. Motif kernel generated by GP improves remote homology and fold detection. *BMC Bioinformatics*, 8(23).
8. John R. Koza. *Genetic Programming*. MIT press, 1992.
9. W. B. Langdon. Evolving GeneChip correlation predictors on parallel graphics hardware. In *WCCI*, Hong Kong, 1-6 June 2008, pages 4152–4157. IEEE.
10. W. B. Langdon and S. J. Barrett. GP in data mining for drug discovery. In A. Ghosh *et al.*, eds., *Evolutionary Computing in Data Mining*, pp211–235. 2004.
11. W. B. Langdon, R. da Silva Camargo, and A. P. Harrison. Spatial defects in 5896 HG-U133A GeneChips. In Joaquin Dopazo *et al.*, eds., *CAMDA 2007*.
12. W. B. Langdon and A. P. Harrison. A grammar based strongly typed genetic programming system for finding regular expression which predict affymetrix DNA probe performance. Technical report, CES-483, University of Essex, UK, 2008.
13. W. B. Langdon, G. J. G. Upton, R. da Silva Camargo, and A. P. Harrison. A survey of spatial defects in Homo Sapiens Affymetrix GeneChips. Submitted.
14. William B. Langdon. *Genetic Programming and Data Structures*. Kluwer, 1998.
15. William B. Langdon and Wolfgang Banzhaf. Repeated sequences in linear genetic programming genomes. *Complex Systems*, 15(4):285–306, 2005.
16. William B. Langdon and Bernard F. Buxton. Evolving receiver operating characteristics for data fusion. In J. F. Miller *et al.*, eds., *EuroGP'2001*, pp87–96.
17. R. I. McKay, T. H. Hoang, D. L. Essam, and Xuan Hoai Nguyen. Developmental evaluation in GP. In P. Collet *et al.*, eds., *EuroGP 2006*, LNCS 3905, pp280–289.
18. D. J. Montana. Strongly typed GP. *Evolutionary Computation*, 3(2):199–230.
19. F. Naef, H. Wijnen, and M. Magnasco. Reply to "comment on 'solving the riddle of the bright mismatches: ' ". *Physical Review E*, 73(6):063902, 2006.
20. Nikolay I. Nikolaev and Vanio Slavov. Concepts of inductive genetic programming. In Wolfgang Banzhaf *et al.*, eds., *EuroGP-98*, LNCS 1391, pp49–60. Springer.
21. M. O'Neill and C. Ryan. Grammatical evolution. *IEEE TEC*, 5(4):349–358, 2001.
22. R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).
23. N. J. Radcliff. Genetic set recombination. *FOGA 2*, pp203–219. Morgan Kaufmann.
24. Brian J. Ross. The evaluation of a stochastic regular motif language for protein sequences. In Lee Spector *et al.*, eds., *GECCO-2001*, pp120–128.
25. Graham JG Upton, William B Langdon, and Andrew P Harrison. Incorrect measurement of gene expression by microarrays. Submitted.
26. P. A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza *et al.*, eds., *Genetic Programming 1996*, pp230–237. MIT Press.
27. P. A. Whigham and P. F. Crapper. Time series modelling using GP: In rainfall-runoff models. In L. Spector *et al.*, eds., *AiGP3*, pp89–104. MIT Press, 1999.
28. M. L. Wong and K. S. Leung. Evolving recursive functions for the even-parity problem using genetic programming. In *AiGP 2*, pp221–240. MIT Press, 1996.